PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Research

Article submitted to journal

Subject Areas:
Artificial Intelligence
Machine Learning

Keywords:
Tsetlin Machine, Regression Tsetlin Machine, Tsetlin Automata, Regression Problems, Pattern Recognition with Propositional Logic, Artificial Intelligence, Machine Learning, Interpretable Machine Learning.

Author for correspondence:
K. Darshana Abeyrathna
e-mail: darshana.abeyrathna@uia.no

THE ROYAL SOCIETY PUBLISHING

The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression

K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin

Centre for Artificial Intelligence Research, University of Agder, Grimstad, Norway.

Relying simply on bitwise operators, the recently introduced *Tsetlin Machine* (TM) has provided competitive pattern classification accuracy in several benchmarks, including text understanding. In this paper, we introduce the *Regression Tsetlin Machine* (RTM), a new class of TMs designed for continuous input and output, targeting non-linear regression problems. In all brevity, we convert continuous input into a binary representation based on thresholding, and transform the propositional formula formed by the TM into an aggregated continuous output. Our empirical comparison of the RTM with state-of-the-art regression techniques reveals either superior or on par accuracy on five datasets.

1. Introduction

Over the last few years, there has been tremendous progress in research on predictive models, resulting in increasingly higher predictive accuracy. However, high predictive accuracy is not always sufficient to trust a predictive model. For applications where humans are making the final decision, the rationale *behind* a prediction can sometimes be essential to the decision making process. Therefore, human interpretability of results is of great importance [1]. This importance has been emphasized in many application domains such as credit scoring [2,3], medicine [4,5], bioinformatics [6,7], and churn prediction [8,9].

The Tsetlin Machine (TM) is a recent machine learning approach that attempts to bridge the gap between the interpretability of rule-based techniques and the high predictive accuracy of deep learning [10].

The TM is relatively simple computationally, being based on propositional logic to form the classifier, and straightforward bitwise operations for recognition and learning. This structure makes the Tsetlin Machine interpretable, yet it achieves competitive classification accuracy for many pattern recognition problems.

The inputs and outputs of the TM are propositional variables, represented as bits. The inputs and their negations, which we call "literals", form conjunctive clauses at the first layer of the TM. In the second layer, classification is performed by a disjunction of all the generated clauses. The number of clauses at the first level is specified by the user. For learning patterns, each clause is assigned one dedicated Tsetlin Automaton (TA) [11] per literal. The decision whether to exclude or include a literal in a particular clause is thus made by the attached TA. The Tsetlin Automata (TAs) that are employed by the TM are thus two-action TA, deciding between an "Exclude"- and an "Include" action. TAs interact with the environment in an iterative manner, the next move of a TA is determined by its current state, which, in turn, is influenced by the feedback it receives from the attached environment [12]. The TAs taking part in the TM is guided by a novel game, with each TA trying to learn the optimal action after a series of interactions with the environment.

There are also other forms of TAs that have been employed to solve challenging problems. These include forecasting disease outbreaks [13], graph coloring [14], distributed coordination [15], stochastic searching on the line [16], and resource allocation [17].

Despite its simplicity, the TM has outperformed traditional machine learning techniques including Neural Networks, Support Vector Machines (SVMs), Logistic Regression, and Naïve Bayes in well-known benchmarks, such as, handwritten digits classification (MNIST), Iris data classification, classification of Noisy XOR data with non-informative features, and predicting optimal moves in the Axis and Allies Board Game [10].

Recently, Berge et al. studied the ability of the TM to categorize text simply based on a document word presence vector [18]. They further investigated the interpretability of the produced rules (clauses), for analyzing electronic health records. In all brevity, they showed that the TM could outperform several vanilla machine learning techniques on text categorization, including SVMs, Decision Trees, Random Forest, k-Nearest Neighbor classifiers (kNN), Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM) Neural Networks, while keeping the important property of interpretability.

Paper Contributions: The classic TM has been designed for binary input patterns and binary output. Although continuous input and output can be encoded in bit form, the inherent properties of continuous values are lost, such as increasing semantic similarity between numerically closer values. We address this limitation in the present paper by introducing the Regression Tsetlin Machine, a novel Tsetlin Machine architecture. For continuous input, we introduce a data preprocessing procedure that transforms the input losslessly into a binary representation that maintains semantic relationship between numbers. That is, the preprocessing procedure considers each unique data sample in each continuous feature as a potential threshold. Then, the original data samples are compared with the complete set of thresholds to create a new feature matrix which only contains bits.

To produce continuous output that leverage the natural ordering of numbers, the we modify the inner inference mechanism of the TM so that input patterns are transformed into a single continuous output, rather than to distinct categories. We achieve this by eliminating the polarities in the clauses, and by summing all the non-polarized votes, mapping the sum into a continuous output. We coin the resulting scheme, the Regression Tsetlin Machine (RTM).

Finally, we propose a new feedback scheme for guiding the TA. We have tailored this scheme for the regression procedure described above. That is, the scheme minimizes the discrepancy between the predicted and the target outputs, with the aid of a modified stochastic clause activation function.

Paper Organization: The remainder of the paper is organized as follows. In Section 2, we discuss the structure and inference mechanisms of the TM, with a particular focus on the parts of the architecture that we build upon. Section 3 and 4, we present the main contribution of

this paper, which is the data preprocessing and regression procedure, leading to the RTM. The behavior of the RTM is then explored in Section 5 by the help of an artificial dataset. Section 6 evaluates the RTM empirically on five real-life datasets, and compared with selected state-of-the-art regression techniques. We conclude our work in Section 7.

2. The Tsetlin Machine (TM)

In classification problems, a class can be represented by its constituting sub-patterns. The TM has been designed to capture these sub-patterns in an effective way. Briefly stated, each clause of the TM captures a specific sub-patterns by means of a conjunction of literals. Accordingly, each class is represented using a series of m clauses, how many decided by the user. The class predicted for a specific input pattern, then, is decided simply by identifying the class with the largest number of matching sub-patterns.

TM structure: Consider an input feature vector $\mathbf{X} = (x_k) \in \{0,1\}^o$ consisting of o propositional variables x_k with domain $\{0,1\}$. The TM considers both the features x_k themselves as well as their negations $\neg x_k$ (jointly referred to as literals) when forming the clauses. In all brevity, each clause c_j takes the following form:

$$c_j = 1 \wedge \left(\bigwedge_{k \in I_j^I} x_k \right) \wedge \left(\bigwedge_{k \in \bar{I}_j^I} \neg x_k \right). \tag{2.1}$$

Above, I_j^I and $\bar{I}_j^I$ are non-overlapping subsets of the input variable indexes, $I_j^I, \bar{I}_j^I \subseteq \{1,\ldots,o\}$, $I_j^I \cap \bar{I}_j^I = \emptyset$, that specify which of the literals take part in the clause c_j. The upper index I denotes that the literals in the set are *Included* in the clause. Conversely, I^E specifies the literals that are *Excluded*. The indexes of the included original features are indicated by I_j^I while the indexes of the included negated features are indicated by $\bar{I}_j^I$.

To simplify notation, we now introduce an augmented feature vector $\mathbf{X}'$, which can be written as $\mathbf{X}' = [x_1, x_2, x_3, \ldots, x_o, \neg x_1, \neg x_2, \neg x_3, \ldots, \neg x_o]$, after concatenating the negated features with the original features. The elements of the augmented feature vector x'_k are now the literals, and we then only need to consider I_j^I and I_j^E, however, for the expanded feature index set $\{1,\ldots,2o\}$.

The TM employs two-action TAs to decide which feature indexes $1,\ldots,2o$ goes into I_j^I, one team of TAs per clause c_j. That is, when there are o features, we need $2 \times o$ TAs per clause. Half of them represent the original features and the remaining represent the negated features.

The two actions available to each TA are *include* and *exclude*. Here, *include* refers to including the literal assigned to the TA and *exclude* means excluding it. The action that the TA performs out of the two available is decided by its current state, $a_{j,k} \in \{1,\ldots,2N\}$. If the state is less than or equal to N, the literal is excluded from the clause. Therefore, the subset of indexes of the excluded literals can be stated as, $I_j^E = \{k | a_{j,k} \leq N, 1 \leq k \leq 2o\}$. Oppositely, if the state is higher than N, the corresponding literal is inserted in the clause. The subset of the indexes of the included literals can be then written as $I_j^I = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$. All these states $a_{j,k}$ of all the clauses are organized in an $m \times 2o$ matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1,\ldots,2N\}^{m \times 2o}$.

Fig. 1 illustrates the above described TM structure. In the figure, the upper index t of TA_k^t denotes the type of the feature (1 for the original and 2 for the negated) while the lower index k denotes the original feature index.

Clause output: Since the TM clause is a conjunction of literals, including any literal of value 0, even a single one, will make the clause output 0. Let the index set $I_{X'}^1$ be the indexes k of all of the literals of value $x'_k = 1$ in the input $\mathbf{X}'$. Accordingly, a clause evaluates to 1 if and only if the indexes of the included literals I_j^I is a subset of the indexes $I_{X'}^1$ of literals of value 1:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j^I \subseteq I_{X'}^1, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

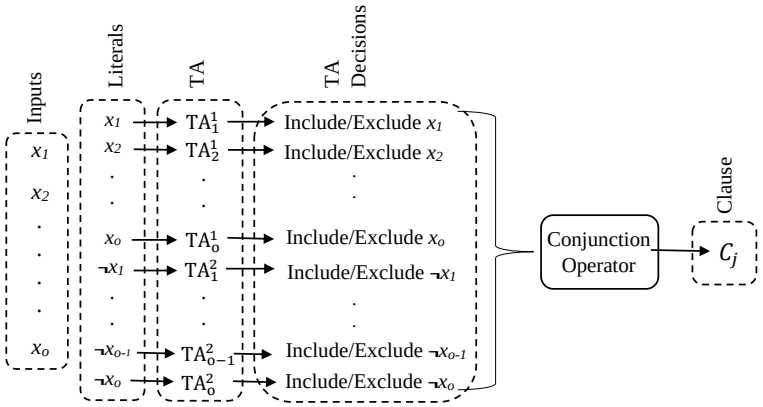

Figure 1: Forming a clause using input features and the actions of the TAs.

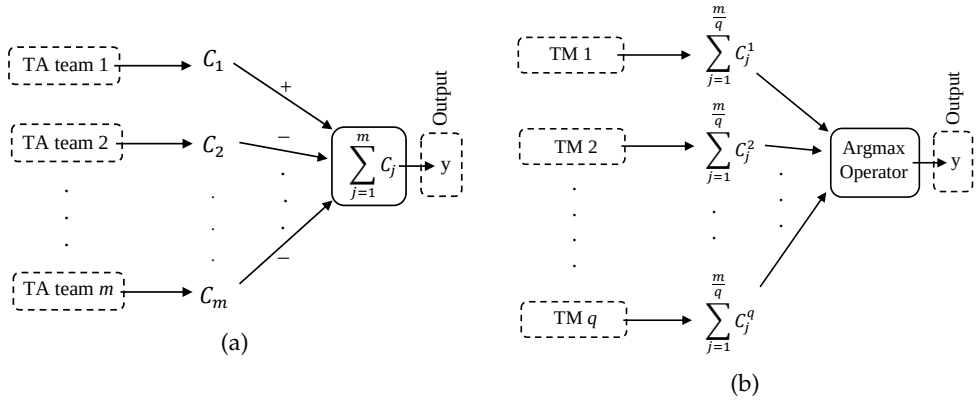

Figure 2: (a) The Basic TM for two class problems. (b) Multi-class version of the TM.

Clause voting: The above clause outputs are then organized in a vector $\mathbf{C} = (c_j) \in \{0,1\}^m$. The number of clauses m should at least be sufficient to cover the full range of sub-patterns associated with each output $\{0,1\}$. When the number of sub-patterns in each class is unknown, a grid search is required to estimate the necessary amount of clauses. As specified earlier, the final output $\{0,1\}$ is selected based on the number of identified sub-patterns (votes) for each class.

To increase the expressiveness of the TM, clauses are assigned positive or negative polarity. Clauses with positive polarity (C^+) vote for the class 1 ($y = 1$) and clauses with negative polarity (C^-) vote for the class 0 ($y = 0$). To avoid bias, the clauses are equally divided among class 0 and class 1. E.g., clauses with odd indexes can be assigned positive polarity and clauses with even indexes can be assigned negative polarity. The TA state matrix $\mathbf{A}$ then has two separate sections: $\mathbf{A}^+$ and $\mathbf{A}^-$. The section $\mathbf{A}^+$ maintains the states of TAs of positive clauses: $\mathbf{A}^+ = (a^+_{j,k}) \in \{1,\dots,2N\}^{\frac{m}{2}\times 2o}$, representing the odd rows of matrix $\mathbf{A}$ and $\mathbf{A}^-$ maintains the states of TAs of negative clauses: $\mathbf{A}^- = (a^-_{j,k}) \in \{1,\dots,2N\}^{\frac{m}{2}\times 2o}$, representing the even rows of matrix $\mathbf{A}$.

The resulting two-class architecture is illustrated in Fig. 2a. The summation operator at the end sums the votes for each class separately and consider the difference in order to decide the final output: $v = \sum_j c^+_j - \sum_j c^-_j$. Then the final output is decided as in (2.3).

$$y = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0\,. \end{cases} \tag{2.3}$$

However, for categorization tasks with more classes than two, separate TM are needed for each class as shown by Fig. 2b. In such situations, clauses are partitioned equally among the classes. Clauses with positive polarity vote in favor of the considered class, and clauses with negative polarity vote against the considered class. That is, an argmax operator arbitrates the class predicted for the input, based on the votes collected for each class. When there are q classes and v^i is the difference between positive and negative votes for class i, the output, y, can be expressed as:

$$y = \text{argmax}_{i=1,\ldots,q}\{v^i\}. \tag{2.4}$$

Learning procedure: The TMs produces the final classifier in Disjunctive Normal Form (DNF). The corresponding conjunctive clauses are formed by the decisions made by the collective of TA teams associated with all of the clauses. Hence, training in the TM entails careful guiding of the TAs so that they as a collective make the optimal *include* and *exclude* decisions. We achieve this by reinforcement learning, organized as a game between the TAs. In all brevity, the TAs adjust their states based on feedback from the game (environment), encompassing rewards, penalties, and inaction feedback. To achieve the learning goal, the probabilities of receiving these different feedback has been designed to account for critical factors, namely, the actual output, the clause outputs, the literal values, and current state of the TAs.

In contrast to gradient-based learning, learning in TM naturally combats false positives and false negatives. As a consequence, it eludes the issues attached to gradient-based algorithms such as vanishing/exploding gradient. In the case of categorization problems with two classes, the basic idea is to penalize voters when they vote to procure a false positive or false negative and to reward voters when they vote to procure a true positive or true negative. In the TM, this is done by two types of feedback – Type I and Type II.

Type I feedback: The clauses that receive Type I feedback is decided stochastically using the following conditions.

$$p_{j,y} = \begin{cases} 1 & \text{with probability } \frac{T-\max(-T,\min(T,v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

Here, the lower indexes j and y of p represent the clause and target class, respectively. A user defined target, T decides the robustness of the learning. Higher T makes more clauses involved in learning which subsequently learn more patterns. However, it has to be decided carefully without exposing the TM to learn noise (overfitting).

Type I feedback is given to clauses with positive polarity when the actual output, $\hat{y}$, is 1 and to clauses with negative polarity when the actual output, $\hat{y}$, is 0. Therefore, the complete matrix, $\mathbf{P}_y$, that picks the clauses for feedback Type I, is a combination of $\mathbf{P}_1^+$ and $\mathbf{P}_0^-$ where $\mathbf{P}_1^+ = (p_{j,1}^+) \in \{0,1\}^{\frac{m}{2}}$ and $\mathbf{P}_0^- = (p_{j,0}^-) \in \{0,1\}^{\frac{m}{2}}$. Type I feedback does two jobs. Type Ia reinforces *include* actions of TAs whose corresponding literal value is 1 when the clause output is 1. Type Ib combats over-fitting by reinforcing *exclude* actions of TAs when the corresponding literal is 0 or when the clause output is 0. Both Ia and Ib feedback are provided to TAs stochastically using a user set parameter s (s ≥ 1).

Type Ia feedback: The matrix $\mathbf{R}$, i.e, $\mathbf{R} = (r_{j,k}) \in \{0,1\}^{m \times 2o}$ stores the decisions whether the k^{th} TA of the j^{th} clause will receive the feedback type Ia. The decisions are made stochastically as follows.

$$r_{j,k} = \begin{cases} 1 & \text{with probability } \frac{s-1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

Now, using the complete set of conditions, the indexes of the TAs which receive the Type Ia feedback I_C^{Ia} can be identified as, $I_C^{\text{Ia}} = \left\{(j,k) | x'_k = 1 \wedge c_j = 1 \wedge p_{j,y} = 1 \wedge r_{j,k} = 1\right\}$.

Type Ib feedback: Similarly, TAs to receive the Type Ib feedback are decided stochastically and decisions are stored in matrix $\mathbf{Q}$, i.e, $\mathbf{Q} = (q_{j,k}) \in \{0,1\}^{m \times 2o}$. In this situation, the decision for the k^{th} TA of the j^{th} clause is,

$$q_{j,k} = \begin{cases} 1 & \text{with probability } \frac{1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Considering all other conditions, the indexes of the TAs that receive the feedback type Ib, I_C^{Ib} are expressed as, $I_C^{\text{Ib}} = \left\{(j,k) | (x'_k = 0 \vee c_j = 0) \wedge p_{j,y} = 1 \wedge q_{j,k} = 1\right\}$.

The identified TAs are updated by changing their internal state. The available update operations are $\oplus$ and $\ominus$. The operator $\oplus$ adds 1 to the current state while operator $\ominus$ subtracts 1 from the current state within the given state space: $\mathbf{A} \leftarrow \left(\mathbf{A} \oplus I_C^{\text{Ia}}\right) \ominus I_C^{\text{Ib}}$

Therefore, Type I feedback compels clauses to output 1 regardless of its polarity (positive or negative). As mentioned earlier, this feedback is applied on clauses with positive polarity when the actual output, $\hat{y}$, is 1. Hence, Type I feedback reinforces the true positive outputs. Furthermore, when the actual output, $\hat{y}$, is 0, Type I feedback is applied on clauses with negative polarity. Correspondingly, Type I feedback also reinforces the true negative outputs.

Type II feedback: Type II feedback is given to clauses with positive polarity when the actual output, $\hat{y}$, is 0 and clauses with negative polarity when the actual output, $\hat{y}$, is 1. The j^{th} clause that receives the Type II feedback based on the target class y is then decided as follows:

$$p_{j,y} = \begin{cases} 1 & \text{with probability } \frac{T + \max(-T, \min(T, v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

Similar to the explanations in Type I feedback, T is the user configured threshold and decisions are stored in the matrix $\mathbf{P}_y$. The idea is to alter the clause output of the clauses which output 1. This can be done by simply including literals which have value 0 in the clauses which output 1. The TAs that match the above conditions can be then identified as, $I_C^{\text{II}} = \left\{(j,k) | x'_k = 0 \wedge c_j = 1 \wedge p_{j,y} = 1\right\}$.

The identified TAs are updated by changing their internal state: $\mathbf{A}^+ \leftarrow \mathbf{A}^+ \oplus I_{C+}^{\text{II}}$. In this way, Type II feedback compels clauses to output 0 regardless of its polarity (positive or negative). Hence, when Type II feedback is applied on clauses with positive polarity when actual output, $\hat{y}$, is 0, it combats false positives. Likewise, when Type II feedback is applied on clauses with negative polarity when actual output, $\hat{y}$, is 1, it combats false negatives.

The training procedure of the multi-class version of the TM is similar to the above training procedure. Clauses of the class being the target of the current training sample are treated as if $\hat{y} = 1$, while the clauses of a randomly selected class from the remaining classes are treated as if $\hat{y} = 0$. In each class, clauses with positive polarity vote to say that the output belongs to the considered class. Similarly, the clauses with negative polarity vote to indicate that the output does not belong to the considered class.

3. An Encoding Scheme for Continuous Input to the Tsetlin Machine

The input feature vector for the TM we discussed in the previous section had only binary features, i.e., $\boldsymbol{X} = [x_1, x_2, x_3, \ldots, x_o]$ with $x_k, k = 1, 2, ..., o$, being 0 or 1. Clauses are directly composed of these features or their negations conditioned on the decisions made by the TAs. However, in many real-world applications, binary features cannot be expected. Instead, the input features are often continuous valued. In order to cope with such applications, we introduce a preprocessing procedure that transforms continuous features into binary variables, while maintaining ranking relationships among the continuous feature values.

Table 1 is an example that has two continuous features, they are respectively listed in Column 1 and Column 2 in the table. The preprocessing procedure follows the following steps to convert them into binary form, one feature at a time.

(i) Identify the unique values $\{v_1, v_2, \ldots, v_u\}$ of the feature.

Table 1: Type I and Type II feedback designed to eliminate false negative and false positive output.

Raw Feature		Thresholds						
1	2	≤ 3.834	≤ 5.779	≤ 10.008	≤ 11.6	≤ 25.7	≤ 32.4	≤ 56.1
5.779	25.7	0	1	1	0	1	1	1
10.008	56.1	0	0	1	0	0	0	1
5.779	11.6	0	1	1	1	1	1	1
3.834	32.4	1	1	1	0	0	1	1

(ii) Sort the identified unique values from smallest to largest.

(iii) Consider each unique value $v_i, i = 1, 2, ..., u$, as a threshold "$\leq v_i$", as shown in sorted order in the "Thresholds" row in the table.

(iv) Compare each original continuous value in the feature with each of the sorted thresholds. If the feature value is greater than the threshold, set the corresponding binary variable to 0, otherwise, set the bit to 1.

(v) Repeat steps (i) to (iv) until all the continuous values are converted into binary form.

According to above the procedure, the first feature in the given example has three unique values, i.e., 5.779, 10.008, and 3.834 (step (i)). These unique values are sorted and considered as thresholds, ≤ 3.834, ≤ 5.779, and ≤ 10.008 (step (ii) and (iii)). Notice that the number of binary feature bits we are going to have after converting the continuous inputs, equals the number of thresholds. In this case, for Feature 1 in the table (first column), three new binary feature bits are introduced, each of which corresponds to its respective threshold. If we compare the first raw continuous value of Feature 1, namely, 5.779, with the identified three thresholds, we get a "011" as the raw value is greater than 3.834 (resulting in 0), equal to 5.779 (resulting in 1), and less than 10.008 (resulting in 1) (Step iv). Similarly, the remaining raw continuous values, 10.008 and 3.834, are converted into 001 and 111, respectively.

Once all the raw continuous values of the first feature are converted, conversion of the second feature can start. This procedure is iterated until all the continuous values in all the continuous features have been converted to binary form (step (v)).

The new binary representation of continuous features becomes particularly powerful because it allows the TM to reason about the ordering of the values, forming conjunctive clauses that specify rules based on thresholds, and with negated features, also rules based on intervals.

4. The Regression Tsetlin Machine (RTM)

For many real-world prediction problems, the target y is continuous valued, referred to as regression problems. Although continuous output can be encoded in bit form, the inherent properties of continuous values are lost, such as increasing semantic similarity between numerically closer values. Thus, the binary output of the classic TM shown in Fig. 2a and Fig. 2b is not ideal for representing continuous output. In order to address this problem, we here propose a new type of TM — the *Regression Tsetlin Machine (RTM)*.

The structure of the RTM is illustrated in Fig. 4. As the figure shows, we removed the polarity of the clauses in the traditional TM structure, instead, we use clauses as additive building blocks to calculate the continuous outputs. That is, the summation operator in Fig. 4 counts the total number of clauses that evaluate to 1, and the resulting sum is further mapped into a continuous value according to Eq. (4.1):

$$y_o = \frac{\sum_{j=1}^{m} C_j(\hat{X}_o) \times (\hat{y}_{\max} - \hat{y}_{\min})}{T} + \hat{y}_{\min}\,, \tag{4.1}$$

where T is the user specified output resolution, $\hat{y}_{\max}$ and $\hat{y}_{\min}$ are respectively the maximum and minimum values of the target output of the training data. We denote the target output of the training data as $\boldsymbol{Y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_M]$, if there are M samples being used for training.

Once a continuous output is calculated, it can be compared with the target output to get the difference between them. Following the convention of regression analysis, we call this difference "error". The output itself and the error are, in turn, fed back to the RTM, and influence the decision made by the TAs so that the error can be minimized. By this procedure of "regression", the goal is for the RTM to learn patterns appropriate for regression from the training dataset.

TAs in the RTM follow the Type I and Type II feedback specified earlier in the paer to update their states. Whether it follows Type I or the Type II feedback, is determined by Eq. (4.2):

$$Feedback = \begin{cases} \text{Type I}, & \text{if} \quad y_o < \hat{y}_o \,, \\ \\ \text{Type II}, & \text{if} \quad y_o > \hat{y}_o \,. \end{cases} \tag{4.2}$$

The idea behind Eq. (4.2) is similar to the rationale behind traditional TM. I.e., we want to increase the number of clauses that output 1 when the predicted output is less than the target output ($y_o < \hat{y}_o$). To achieve this, Type I feedback is introduced. Conversely, Type II feedback is applied to decrease the number of clauses that evaluate to 1 when the predicted output is higher than the target output ($y_o > \hat{y}_o$).

If the feedback determined by (4.2) is offered to all the clauses, the predicted value will keep oscillating around the target output throughout the training process. Hence, we suppress the number of clauses that receive the feedback so that the fraction of clauses receiving feedback is proportional to the error between the predicted and the target output. This can be achieved by introducing the following feedback activation probability, P_{act}, by which the RTM decides stochastically the number of clauses to be updated.

$$P_{act} = \frac{K \times \mid y_o - \hat{y}_o \mid}{\hat{y}_{\max} - \hat{y}_{\min}} \,. \tag{4.3}$$

Note that in Eq. (4.3), the parameter K is added to adjust the activation probability according to the total number of clauses. In other words, the probability varies proportionally to the error, and is reasonably scaled to fit the total number of clauses.

The flowchart given in Fig. 4 summarizes the complete training procedure of the RTM, including the data preprocessing step introduced in Section 3. The trained RTM is able to utilize the learned pattern in conjunctive clauses to predict unseen data.

5. The Behavior of the Regression Tsetlin Machine

We study the behavior of the RTM by the means of two artificial datasets. The two datasets are basically the same, except that Dataset II is noisy and Dataset I is noise free [1].

[1]The artificial datasets and RTM code are available to download at https://github.com/cair/regression-tsetlin-machine

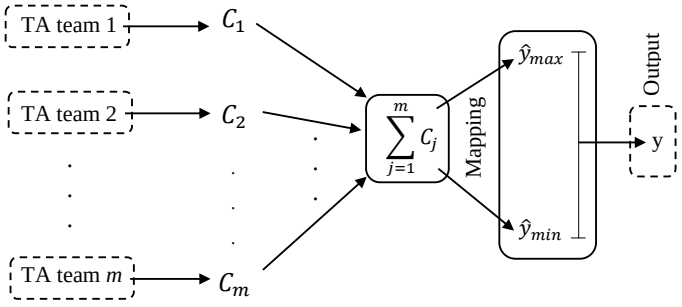

Figure 3: The RTM structure.

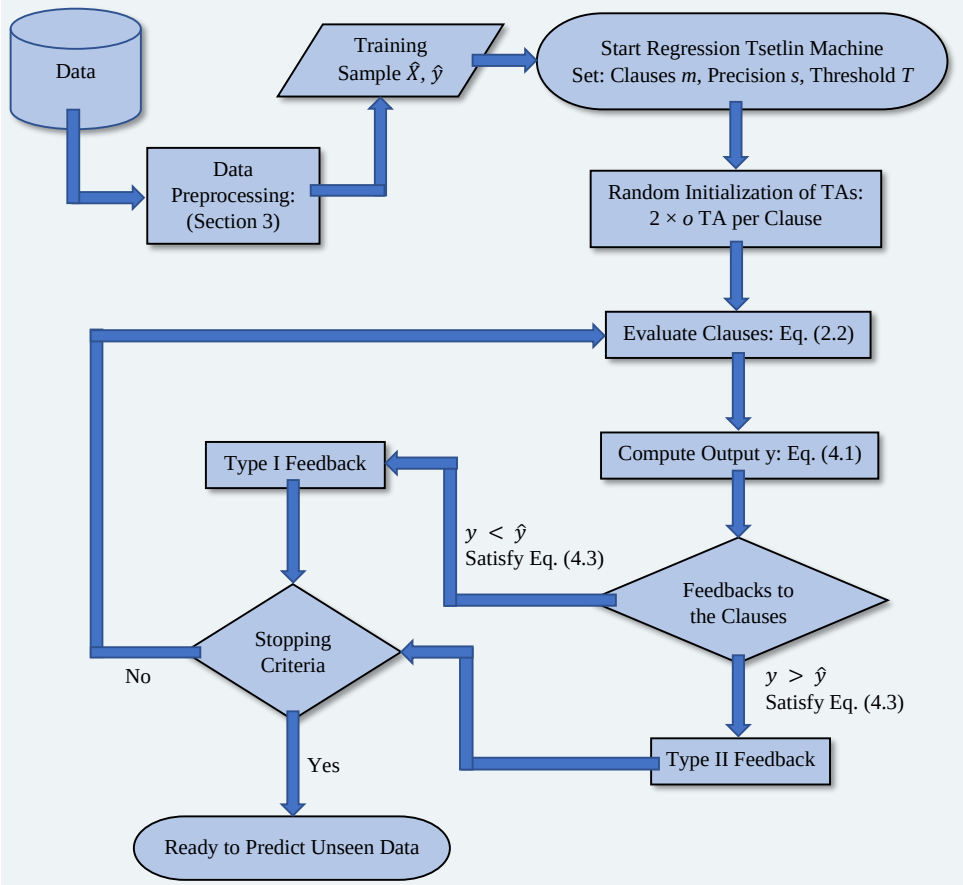

Figure 4: The training work-flow of the RTM.

(a) Artificial Data

In the noise free Dataset I, as shown in Table 2, there are 8 distinct outputs, each of which is triggered by a specific 3-bit input. Each feature bit has an equal probability of being 0 or 1, leading to an uniform distribution of bit values. The continuous output considered in this experiment is 100 times the decimal representation of the binary input. For example, the continuous output for the input (011) is 100 times its decimal output, i.e., $100 \times 3 = 300$. The complete dataset consists of 10,000 samples. A big portion (80%) of the entire dataset is utilized in training the RTM and the rest (20%) is used to examine the performance of the RTM after training.

Dataset II contains the same data as Dataset I, except that the output of the training data is perturbed to introduce noise.

Table 2: The Artificial Data without Noise.

Feature			Decimal	Noise Free
1	2	3	Output	Continuous Outputs
0	0	0	0	0
0	0	1	1	100
0	1	0	2	200
0	1	1	3	300
1	0	0	4	400
1	0	1	5	500
1	1	0	6	600
1	1	1	7	700

(b) Behavior Analysis

We train and test the RTM separately on the above two datasets. The Mean Absolute Error (MAE) is utilized as the criterion to evaluate the performance of the RTM. When there are G samples in the predicted and actual series, the MAE can be calculated as,

$$MAE = \frac{\sum_{g=1}^{G} | y_g - \hat{y}_g |}{G}. \tag{5.1}$$

Fig. 5 and Fig. 6 depict the variation of the training MAE across 1000 epochs during the training process. Each figure demonstrates five curves in five different colors, representing the results obtained by adopting five different resolutions T. The final training MAEs, i.e., the ones obtained at the end of the training process, and testing MAEs, are also given in the parentheses in the legends.

Fig 5 manifests that noise-free data can be perfectly learned (MAE = 0.00) by the RTM with merely 7 clauses. The underlying idea can be enumerated as follows:
- If TAs in any *four* of the seven clauses have made the following decisions:
$\text{TA}_1^1 = \{include\}$, $\text{TA}_1^2 = \{exclude\}$, $\text{TA}_2^1 = \{exclude\}$, $\text{TA}_2^2 = \{exclude\}$, $\text{TA}_3^1 = \{exclude\}$, and $\text{TA}_3^2 = \{exclude\}$,
- and if TAs in any *two* of the seven clauses made the decisions:
$\text{TA}_1^1 = \{exclude\}$, $\text{TA}_1^2 = \{exclude\}$, $\text{TA}_2^1 = \{include\}$, $\text{TA}_2^2 = \{exclude\}$, $\text{TA}_3^1 = \{exclude\}$, and $\text{TA}_3^2 = \{exclude\}$,
- and if the TAs in the remaining *one* clause have decided:

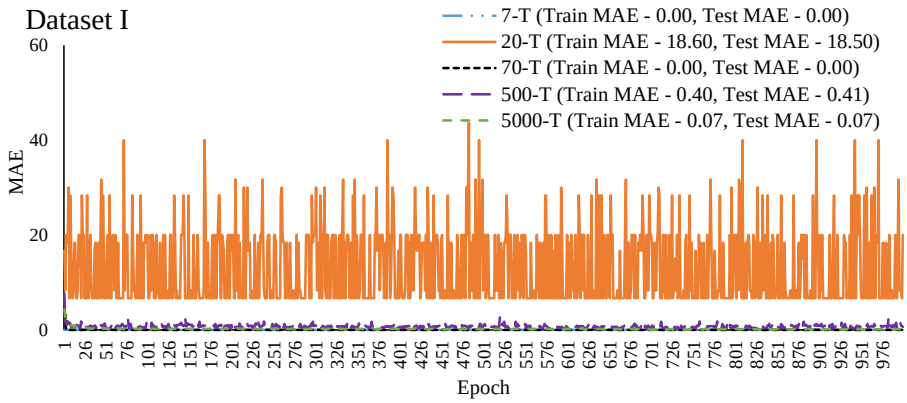

Figure 5: Training error variation for Dataset I. The dataset is processed with different T.

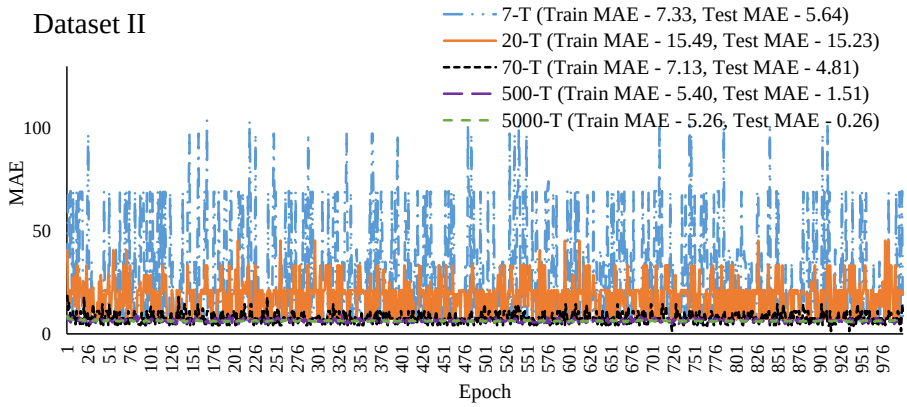

Figure 6: Training error variation for Dataset II. The dataset is processed with different T.

$\text{TA}_1^1 = \{exclude\}$, $\text{TA}_1^2 = \{exclude\}$, $\text{TA}_2^1 = \{exclude\}$, $\text{TA}_2^2 = \{exclude\}$, $\text{TA}_3^1 = \{include\}$, and $\text{TA}_3^2 = \{exclude\}$,
then, these clauses can simply compute any value in the target series collectively, as the first four clauses capture the pattern $(1 * *)$, the next two clauses capture the pattern $(* 1 *)$, and the last clause captures the pattern $(* * 1)$. Here, $*$ means the input feature that can take a value from either 0 or 1. In other words, the value of $*$ does not really matter since both the TA attached to the original input and the TA associated with its negation decide to *exclude* the literal from the clause.

We take the input (010) as an example to explain this in detail. (010) does not activate the first set of clauses, as the clauses are explicitly $\{include(0) \wedge exclude(\neg 0) \wedge exclude(1) \wedge exclude(\neg 1) \wedge exclude(0) \wedge exclude(\neg 0)\}$, which results in 0. Similarly, (010) does not activate the clause which represents the pattern $(* * 1)$. However, the input (010) indeed activates the second set of clauses since TA_2^1 *include* the second feature bit (1), and all the other TAs choose to *exclude* their literals, which results in the clause value being 1. Furthermore, since the second set has two clauses, Eq. (4.1) guides the RTM to compute the output to be 200.

Likewise:
- Output 100 is computed by activating the clause that represent the pattern $(* * 1)$,
- Output 300 is computed by activating the clauses that represent the patterns $(* 1 *)$ and $(* * 1)$,
- Output 400 is computed by activating the clauses that represent the pattern $(1 * *)$,
- Output 500 is computed by activating the clauses that represent the patterns $(1 * *)$ and $(* * 1)$,
- Output 600 is computed by activating the clauses that represent the patterns $(1 * *)$ and $(* 1 *)$,
- and Output 700 is computed by activating the clauses representing the patterns $(1 * *)$, $(* 1 *)$, and $(* * 1)$.

The same dataset can also be perfectly learned when we set the resolution T to be 70. This is because 70 is a number that equals 7 times an integer, in such cases, clauses can be perfectly divided into distinct groups to identify different patterns. In cases where the exact number of clauses required is unknown, and where the machine is working with noisy data, assigning a relatively large resolution T is advantageous. This is demonstrated in Fig. 5. Here, the training and testing MAEs for $T = 20$ is significantly higher than the training and testing MAEs for $T = 500$ and $T = 5000$ for the Dataset I.

The latter reasoning also applies to the results for Dataset II. I.e., the training and testing MAEs decrease while increasing the resolution T. Besides, the testing MAE can be further reduced towards 0 by increasing the resolution and the number of training epochs.

The s value used in the Type I and Type II feedback is set to be 2 for both datasets in our experiment, considering the results reported in [19]. The reason can be explained with the aid of Fig. 7, where one sees the distribution of patterns when the dataset has 3 feature bits. The distribution can be divided into 8 distinct unique sub-patterns, and the probability of occurrence of any one of these sub-patterns is $\frac{1}{8}$. The shaded area in the figure exhibits the pattern $(1 * *)$,

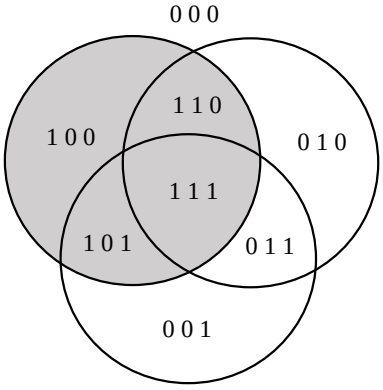

Figure 7: Pattern distribution for the 3-bits input datasets.

whose probability of occurrence is $\frac{4}{8}$. Similarly, the probabilities of occurrence for both the pattern $(* \; 1 \; *)$ and the pattern $(* \; * \; 1)$ are $\frac{4}{8}$, too. According to the Tsetlin Machine dynamics explained in [10], in order to capture a pattern by a clause, $1/s$ should be set equal to the probability of occurrence of that pattern. Therefore, $1/s$ in our experiment is set to $1/2$ (=4/8). However, when the pattern distribution is unknown, we need to do grid search to find a proper s to fit the hidden patterns.

6. The Regression Tsetlin Machine on Real-World Data

Apart from the artificial datasets, we also study the behavior of RTM on five different real-life datasets:

Dengue Incidences:[2] This dataset consists of monthly dengue incidences in the Philippines per 100,000 population. The Philippines has 17 administrative regions. The department of health in the Philippines had collected this data for all the regions separately from 2008 to 2016. In this experiment, the RTM is trained separately for each region using the data from 2008 to 2015. Dengue incidences from the neighboring regions are used as input features. More details about data preprocessing and feature selection can be found in [20]. Dengue incidences in 2016 are used as the testing data.

Energy Performance:[3] In this application, heating load of residential buildings has to be estimated using eight input features: glazing area distribution, glazing area, orientation, relative compactness, wall area, surface area, overall height, roof area, and orientation [21]. The complete dataset comprises 768 samples. 80% of the data samples is utilized for training the model and the rest is used to evaluate it.

Stock Selection:[4] The US historical stock market data had been utilized to simulate the weights of the stock-picking concepts in [22]. These weights of the stock-picking concepts then should be employed to build a stock selection decision support system. The dataset consists of six output variables. Only two of them, i.e., Annual Return and Real Win Rate in this experiment are selected to predict with the RTM. Again, 80% of the total data samples is utilized to train the model and the rest is utilized to test it.

Real Estate Valuation:[5] This is a time series dataset which is to estimate the house price per unit area in Taiwan [23]. The house price per unit area (Dollars per Ping, 1 Ping = 3.3 meter squared) is estimated using six features, namely the transaction date, the house age, the distance to the nearest MRT station, the number of convenience stores in the living circle on foot, and the geographical coordinates (latitude and longitude). A high portion (80%) of the complete dataset is utilized to train the RTM and the rest (20%) is used to examine the performance of the RTM after training.

Aerofoil Noise:[6] The data set comprises different size of NACA 0012 airfoils (an airfoil shape for aircraft wings) at various wind tunnel speeds and angles of attack. The goal is to predict the self-generated noise of an airfoil blade in decibels using five attributes, namely, Frequency (in Hertzs), Angle of attack (in degrees), Chord length (in meters), Free-stream velocity (in meters per second), and Suction side displacement thickness (in meters). Out of the total 1503 data samples, 625 are randomly selected for doing the experiment, and the RTM is trained on 80% of the selected data and tested on the rest of them.

Separate hyperparameter settings are found for different datasets based on grid search. Means of MAEs are obtained after replicating each experiment 20 times and are reported in Table 3 with 95% confidence intervals. To evaluate the performance of the RTM, three other state-of-the-art machine learning models are being compared with. These models are Regression Tree (RT), Random Forest (RF), and Support Vector Regression (SVR), and they are configured with their best parameter settings in order to conduct a fair comparison. Each model predicts six different

[2]The dataset is available to download at https://www.kaggle.com/grosvenpaul/dengue-cases-in-the-philippines
[3]The dataset is available to download at https://archive.ics.uci.edu/ml/datasets/Energy+efficiency
[4]The dataset is available to download at https://archive.ics.uci.edu/ml/datasets/Stock+portfolio+performance
[5]The dataset is available to download at https://archive.ics.uci.edu/ml/datasets/Real+estate+valuation+data+set
[6]The dataset is available to download at https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise#

Table 3: Means of MAEs with 95% confidence intervals by different models on different datasets.

Model	Dengue Incidences	Energy Performance	Stock Selection		Real Estate Valuation	Aerofoil Noise
			Annual Return	Real Win Rate		
RTM	5.160	0.487±0.00	0.078±0.00	0.090±0.00	5.205±0.00	2.206±0.00
RT	7.769	0.598±0.00	0.091±0.00	0.096±0.00	6.033±0.11	2.280±0.07
RF	9.122	0.563±0.00	0.088±0.00	0.097±0.00	5.360±0.00	1.921±0.00
SVR	5.305	1.155±0.00	0.073±0.00	0.092±0.00	5.697±0.00	2.156±0.00

data series using their relevant input features. Out of these six attempts, RTM obtains the best MAE in four of them. For the other two data series, RTM obtains the second and the third best MAEs.

As can be seen in Table 3, the average MAE on forecasting of dengue incidences in all regions in the Philippines shows, that the RTM outperforms the other three models by obtaining the lowest average MAE, which is 33.58%, 43.43%, and 2.73% reduction of the average MAE obtained by RT, RF, and SVR, respectively. In fact, a fixed parameter setting was utilized for RTM for all the regions in the experiment, however, the MAE can be further reduced with different parameter settings for different regions.

The RTM also outperforms other models while predicting the heating load in the Energy Performance dataset, the Real Win Rate in the Stock Selection Dataset, and the house price per unit area in the Real Estate Valuation dataset. While estimating the Annual Return in the Stock Selection dataset, the SVR acquires the lowest MAE, and the RTM is the second best. However, for predicting the self-generated noise in the Aerofoil Noise dataset, RF comes out on top, with SVR being second, closely followed by RTM.

In summery, the RTM outperforms RT in all the six datasets, and when compared with RF and SVR, the RTM obtains better results in five and four of the datasets, respectively.

7. Conclusion

In this paper, we proposed the Regression Tsetlin Machine, a novel variant of the Classic Tsetlin Machine that supports continuous output in non-linear regression problems. We also introduced a data preprocessing procedure which converts continuous inputs into a lossless binary feature matrix. In RTM, the polarities in clauses were removed and the inner inference mechanism was modified so that the input patterns can be transformed into individual continuous outputs, rather than to distinct categories. The number of affected clauses to receive the feedback in RTM was decided stochastically by adopting a linear activation probability function. The behavior of the new algorithm was studied by applying it to both artificial and real-life datasets, where RTM showed superior performance while predicting the Dengue incidences in Philippines, heating load in the Energy Performance dataset, Real Win Rate in the Stock Selection Dataset, and House price per unit area in the Real Estate Valuation dataset compared with RT, RF, and SVM.

References

1. A. A. Freitas, "Comprehensible classification models: a position paper," *ACM SIGKDD explorations newsletter*, vol. 15, no. 1, pp. 1–10, 2014.
2. B. Baesens, C. Mues, M. De Backer, J. Vanthienen, and R. Setiono, "Building intelligent credit scoring systems using decision tables," in *Enterprise Information Systems V*, pp. 131–137, Springer, 2004.
3. J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models," *Decision Support Systems*, vol. 51, no. 1, pp. 141–154, 2011.

4. R. Bellazzi and B. Zupan, "Predictive data mining in clinical medicine: current issues and guidelines," *International journal of medical informatics*, vol. 77, no. 2, pp. 81–97, 2008.
5. M. J. Pazzani, S. Mani, andW. R. Shankle, "Acceptance of rules generated by machine learning among medical experts," *Methods of information in medicine*, vol. 40, no. 05, pp. 380–385, 2001.
6. A. A. Freitas, D. C. Wieser, and R. Apweiler, "On the importance of comprehensible classification models for protein function prediction,"*IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 7, no. 1, pp. 172–182, 2010.
7. D. Szafron, P. Lu, R. Greiner, D. S.Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, A. Fyshe, et al., "Proteome analyst: custom predictions with explanations in a web-based tool for high-throughput proteome annotations," *Nucleic acids research*, vol. 32, no. suppl2, pp. W365–W371, 2004.
8. E. Lima, C. Mues, and B. Baesens, "Domain knowledge integration in data mining using decision tables: Case studies in churn prediction," *Journal of the Operational Research Society*, vol. 60, no. 8, pp. 1096–1106, 2009.
9. W. Verbeke, D. Martens, C. Mues, and B. Baesens, "Building comprehensible customer churn prediction models with advanced rule induction techniques," *Expert systems with applications*, vol. 38, no. 3, pp. 2354–2364, 2011.
10. O.-C. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," *arXiv:1804.01508*.
11. M. L. Tsetlin, "On Behaviour of Finite Automata in Random Medium," *Avtom I Telemekhanika*, vol. 22, pp. 1345–1354, 1961.
12. K. S. Narendra and M. A. Thathachar, Learning Automata: An Introduction. *Courier Corporation*, 2012.
13. K. D. Abeyrathna., O.-C. Granmo, and M. Goodwin, "A Novel Tsetlin Automata Scheme to Forecast Dengue Outbreaks in the Philippines," in *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 680–685, IEEE, 2018.
14. N. Bouhmala and O.-C. Granmo, "Stochastic Learning for SAT-Encoded Graph Coloring Problems," *International Journal of Applied Metaheuristic Computing (IJAMC)*, vol. 1, no. 3, pp. 1–19, 2010.
15. B. Tung and L. Kleinrock, "Using Finite State Automata to Produce Self-Optimization and Self-Control," *IEEE transactions on parallel and distributed systems*, vol. 7, no. 4, pp. 439–448, 1996.
16. B. J. Oommen, S.-W. Kim, M. T. Samuel, and O.-C. Granmo, "A Solution to the Stochastic Point Location Problem in Metalevel Nonstationary Environments," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 2, pp. 466–476, 2008.
17. O.-C. Granmo and B. J. Oommen, "Solving Stochastic Nonlinear Resource Allocation Problems Using a Hierarchy of Twofold Resource Allocation Automata.," *IEEE Transaction on Computers*, 2010.
18. G. T. Berge, O.-C. Granmo, T. Oddbjørn Tveit, M. Goodwin, L. Jiao, and B. Viggo Matheussen, "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization with Medical Applications," *arXiv e-prints, p. arXiv:1809.04547*, Sep 2018.
19. K. D. Abeyrathna, O.-C. Granmo, L. Jiao, and M. Goodwin, "The regression tsetlin machine: A tsetlin machine for continuous output problems," *p. arXiv:1905.04206*, 2019.
20. K. D. Abeyrathna, O.-C. Granmo, X. Zhang, and M. Goodwin, "A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Springer, 2019.
21. A. Tsanas and A. Xifara, "Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools," *Energy and Buildings*, vol. 49, pp. 560–567, 2012.
22. Y.-C. Liu and I.-C. Yeh, "Using mixture design and neural networks to build stock selection decision support systems," *Neural Computing and Applications*, vol. 28, no. 3, pp. 521–535, 2017.
23. I.-C. Yeh and T.-K. Hsu, "Building real estate valuation models with comparative approach through case-based reasoning," *Applied Soft Computing*, vol. 65, pp. 260–271, 2018.